Naval Surface Warfare Center
Carderock Division
West Bethesda, MD 20817-5700

NSWCCD-50-TR–2010/056 August 2010

Hydromechanics Department Report

A Method for Unstructured Mesh-to-Mesh Interpolation

by

William G. Smith
Michael P. Ebert

NSWCCD-50-TR–2010/056 A Method for Unstructured Mesh-to-Mesh Interpolation

This page intentionally left blank.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
15-Aug-2010	Final	-

4. TITLE AND SUBTITLE

A Method for Unstructured Mesh-to-Mesh Interpolation

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER
602123N

6. AUTHOR(S)

William G. Smith

Michael P. Ebert

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) AND ADDRESS(ES)

Naval Surface Warfare Center
Carderock Division
9500 Macarthur Boulevard
West Bethesda, MD 20817-5700

8. PERFORMING ORGANIZATION REPORT NUMBER

NSWCCD-50-TR-2010/056

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

ONR 333
Chief of Naval Research
Ballston Centre Tower One
800 North Quincy Street
Arlington, VA 22217-5660

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

An efficient method for unstructured mesh-to-mesh interpolation is described. This method uses a binary space partitioning tree to sort the elements of the source mesh. Using this tree data structure the source mesh elements can be efficiently searched to find the nearest element for each of the destination mesh points. Once found, the nearest source mesh element is used to compute barycentric coordinates which are then used as weighting coefficients for the interpolation.

15. SUBJECT TERMS

Fluid Structure Interaction, Interpolation, Unstructured, Computational Fluid Dynamics, Binary Space Partition Tree, Barycentric Coordinates

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Michael P. Ebert
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	SAR	9	19b. TELEPHONE NUMBER *(include area code)* 301-227-5403

This page intentionally left blank.

Contents

Figures

This page intentionally left blank.

Abstract

An efficient method for unstructured mesh-to-mesh interpolation is described. This method uses a binary space partitioning tree to sort the elements of the source mesh. Using this tree data structure the source mesh elements can be efficiently searched to find the nearest element for each of the destination mesh points. Once found, the nearest source mesh element is used to compute barycentric coordinates which are then used as weighting coefficients for the interpolation.

Administrative Information

The work described in this report was performed by the Computational Hydromechanics Division (Code 5700) of the Hydromechanics Department at the Naval Surface Warfare Center, Carderock Division (NSWCCD). The work was funded by the Office of Naval Research (Program Officer: Dr. Ki-Han Kim) under the Force Protection Applied Research Program (PE 602123N).

Background

This report describes the code and algorithms used to perform a one way coupling of a Computational Fluid Dynamics (CFD) solver, and a Computational Structural Dynamics (CSD) solver. This code was developed at the Naval Surface Warfare Center, Carderock Division under the FY08 ONR funded Crashback program. Specifically the methods used to transpose the time varying fluid dynamic surface pressure field computed by the CFD solver, to the wetted faces of the finite element mesh used in the CSD solver are described. In this report the CFD surface mesh is referred to as the source mesh, and the CSD mesh as the destination mesh. Transposing the surface pressure field requires that for each point in the destination mesh, a nearest element is found in the source mesh from which all desired scalar, and vector quantities may be interpolated.

Binary Space Partitioning Tree

Due to the relatively large mesh sizes involved (typically greater than $1x10^3$) a Binary Space Partition (BSP) tree, a type of binary search tree, is used to sort the elements of the source mesh. This offers an efficient mechanism for identifying the nearest element in the source mesh to a given point in the destination mesh. Given the large mesh sizes and that each destination mesh point needs an interpolated value, the time invested in building the BSP tree is considered acceptable. The effort involved in building the BSP tree is expected to be on the order of nlog(n) where n is the number of elements in the source mesh. Once the tree is built the time necessary to search for the nearest element is expected to be on the order of log(n) [1].

Building the Tree

For this application a variant of a BSP tree known as a generalized pseudo kd-tree [2] is used. To build the tree the source mesh elements are recursively divided into two bounding

boxes. A bounding box is a rectilinear box aligned with the coordinate axis. It represents the minimum bounding volume of all elements associated with the node of the tree. It could be defined in terms of X_{min}, X_{max}, Y_{min}, Y_{max}, and Z_{min}, Z_{max}, however for simplicity the vectors $P_{min}(X_{min},Y_{min}, Z_{min})$, and $P_{max}(X_{max},Y_{max},Z_{max})$ are defined. Each division is based on cutting the parent node's bounding box across its longest axis, and through the average value of all the element centers. Using the element centers simplifies the division, and sorting of the elements. Each node of the tree contains:

- A pointer to an array of all elements (**Elements*)
- An index to the first element in the tree's node (I_o)
- The number of elements in the node (*NumberOfSubElements*)
- Two vectors which represent a bounding box (P_{min}, and P_{max})
- Pointers to both sub nodes of the tree (S_0, S_1)

The original array of all elements is used and never copied during the process of building the tree. Each sub-node simply points to a sub-section of the array. Elements within the array are than reorganized/sorted by the sub-nodes. The root node of the tree is created by:

- Loading all elements from the source mesh into the element array
- Setting I_o to zero
- Setting *NumberOfSubElements* to the total number of elements in the source mesh
- Looping through all nodes, of each element, and calculating P_{min}, and P_{max}
- Setting the Pointers S_0, and S_1 to Null

The tree is grown by recursively subdividing the root until the *NumberOfSubElements* is 50 or less. Nodes which have 50 or less elements become terminal, or leaf nodes. The procedure for recursively subdividing the root node is:

If and only if the *NumberOfSubElements* is greater than 50:

- Find the longest edge of the bounding box
- Find the average of all the element centers
- Divide the node's bounding box with a plane that cuts orthogonally through the longest edge of the bounding box and through the average of the element centers
- Sort the elements in the current cell so that all elements whose centers are below the plane go in front of all the elements that are not
- Create new S_0:
    - $S_0 \rightarrow I_o = I_o$
    - $S_0 \rightarrow$ *NumberOfSubElements* = Number of elements found below the plane
    - Loop through all nodes, of each element in S_0, to calculate $S_0 \rightarrow P_{min}$, and $S_0 \rightarrow P_{max}$
- Create new S_1:

- $S_1 \rightarrow I_o = S_0 \rightarrow NumberOfElements$
- $S_1 \rightarrow NumberOfSubElements$=Number of elements not found to be below the plane
- Loop through all nodes, of each element in S_1, to calculate the $S_1 \rightarrow P_{min}$, and $S_1 \rightarrow P_{max}$

- Repeat procedure for S_0, and S_1

Searching for the Nearest Element

Once the tree is populated it is used to search for the nearest element in the source mesh to a destination point. Given the coordinates of a destination point, the distance between it and each sub-node's bounding box is calculated, (zero being returned if the point is inside the bounding box). If one has a closer distance it is searched first, otherwise S_1 is arbitrarily searched first. This process is repeated recursively until a terminating node is reached; at this point each element associated with this terminating node is tested individually to find which is closest to the destination point. The index of the closest element, and the distance that it is from the destination point is retained during the search process to eliminate branch searches whose bounding box is farther from the destination point than the current nearest element. For our particular application the source mesh contained only triangular elements.

Finding the Distance to a Triangle

Before accurately computing the distance to a source triangle two preliminary tests are performed in an attempt to minimize the number of triangles that are rigorously tested, thus saving additional expensive calculations. First each destination point will have an average normal calculated for it based on all the destination mesh elements that it is part of. Each triangle is tested to find out if the dot product of the average normal of the destination point and the normal of the source triangle is greater than zero. Considering only source elements for which this dot product is greater than zero ensures that only the source elements that are faced in the same general direction as the surface of the destination point are used for interpolation. The second test is to see if its distance to the plane of the triangle is greater than the distance to the closest triangle so far. If this distance is less than the distance to the current nearest triangle then more rigorous calculations are required to determine the actual distance to the triangle.

To begin calculating the actual distance to the triangle a test is performed to find out if the destination point is over the triangle. This test is performed by considering a tetrahedron with the destination point (P_d) at the peak, and the points to the source triangle (P_0, P_1, P_2) at the base. If the point P_d projects into the base then the distance to the plane of the triangle previously calculated is in fact the distance to the triangle. If P_d projects outside the base then one of the faces will be obtuse to the base, and the distance from the edge to P_d is required. Specifically if the normal defined by source triangle (P_0, P_1, P_2) dotted with the normal defined by triangle (P_d, P_0, P_1) is less than zero then that face of the tetrahedron is obtuse to the base and another calculation will be required to find the distance to the P_0, P_1 edge. Similarly, if the normal defined by triangle (P_0, P_1, P_2) dotted with the normal defined by triangle (P_d, P_1, P_2) is less than zero then another calculation will be required to find the distance to the P_1, P_2 edge. Also, if the normal defined by triangle (P_0, P_1, P_2) dotted with the normal defined by triangle (P_d, P_2, P_0) is

less then zero then the distance to the P_2, P_0 edge is required. Figure 1 contains pseudo code for this method. This pseudo code assumes that VectorType defines the dot product using the "*" operator and the cross product using the "|" operator.

Finding the Distance to an Edge

If during the process of calculating the distance to a source mesh triangle it is found that the destination point cannot be projected onto the triangle then the distance to the closest edge is computed. For example, assuming that the edge P_0, P_1 is the closest edge, the first step is to test

```
Float64 BinaryTreeType::
DistanceFromTheTriangle(const VectorType&  Pd,
                        const VectorType&  PdNormal,
                        const ElementType& Element,
                        SearchResultsType& Results)
    {
    VectorType& P0 = Element.P0
    VectorType& P1 = Element.P1
    VectorType& P2 = Element.P2

    VectorType BaseNormal = NormalToTriangle(P0,P1,P2);

    Float64 d=Abs(BaseNormal*(Pd-P0)); // d=TheDistanceToThePlane

    // Think of a tetrahedron pd at the peak, P0,P1,P2 at the base
    // if the Pd projects into the base then the height will do.
    // if Pd projects outside the base then one of the faces will
    // be obtuse to the base, and the DistanceFromTheEdge will be
    // required.

    if (PdNormal*BaseNormal<0.0) // If normals are not in the same
    if (d<Results.d)             // direction Or if the distance
        {                        // to the plane is more then
                                 // results then don't bother.

        if      (BaseNormal*NormalToTriangle(P0,P1,Pd)<0.0)// If obtuse
            d=DistanceFromTheEdge(Pd,P0,P1);               // use edge
        else if (BaseNormal*NormalToTriangle(P1,P2,Pd)<0.0)
            d=DistanceFromTheEdge(Pd,P1,P2);
        else if (BaseNormal*NormalToTriangle(P2,P0,Pd)<0.0)
            d=DistanceFromTheEdge(Pd,P2,P0);

        if (d<Results.d)       // if distance to the Triangle is
            {                  // less then results then update results
            Results.d    = d;
              Results.Index= Element.Index;
             }
        }
    };
```

Figure 1. Pseudo Code for Finding the Distance to a Triangle

whether or not the triangle P_d, P_0, P_1 is obtuse at P_0, or P_1, if it is then the distance from the obtuse point to P_d is the distance from the edge. Otherwise the height of the triangle is used as the distance to the edge. Figure 2 contains pseudo code for this method.

Interpolation

Once the nearest source triangle to a destination point has been identified any scalar or vector data associated with the points of the source triangle can be interpolated to the destination point. This application uses a variant of barycentric coordinates [3,4]. Barycentric coordinates are weighting coefficients that when multiplied by the source triangle points sum to the destination point. Specifically barycentric coordinates satisfy the following equation,

$$P=C_0*P_0+C_1*P_1+C_2*P_2$$

$$1=C_0+C_1+C_2$$

where P could be any point in the plane of the triangle and P_0, P_1, P_2 are points of the nearest source triangle, and C_0, C_1, and C_2 are the barycentric coordinates. By definition the barycentric coordinates must add up to 1. The barycentric coordinates are used to interpolate any scalar by simply multiplying the coordinates by their respective scalar values like so,

$$S_d=C_0*S_0+C_1*S_1+C_2*S_2$$

Unfortunately, for this application P_d is not likely to lie on the plane of the triangle, so it will be projected to the nearest point in the plane of the triangle. The barycentric coordinates are then calculated using the following equations:

```
Float64 BinaryTreeType::
DistanceFromTheEdge(const VectorType& Pd,
                    const VectorType& P0,
                    const VectorType& P1)
    {

    VectorType L =P1-P0;
    VectorType R0=Pd-P0;
    VectorType R1=Pd-P1;
    Float64    d;

    if      (R0*L<0.0)      // If triangle obtuse at P0
        d=Abs(Pd-P0);       // Use distance From The Pd to P0
    else if (R1*L>0.0)      // If triangle is obtuse at P1
        d=Abs(Pd-P1);       // Use distance From The Pd to P1
    else
        d=Abs(R0|Unit(L)); // Use height of the triangle

    return(d);

    };
```

Figure 2. Pseudo Code for Finding the Distance to an Edge

$$TotalArea = AreaOfTriangle\ (P_0,P_1,P_2)$$

$$C_0 = AreaOfTriangle(P,P_1,P_2)/\ TotalArea$$

$$C_1 = AreaOfTriangle(P,P_2,P_0)/\ TotalArea$$

$$C_2 = AreaOfTriangle(P,P_0,P_1)/\ TotalArea$$

In the current implementation of the code these values are always positive and as such are incorrect if P is outside the boundaries of the source triangle. The correct coordinates would need to have at least one negative value if P was outside the boundaries of the source triangle. Using the barycentric coordinates under these conditions is akin to extrapolation rather than interpolation. In order to avoid this type of extrapolation and to ensure that we don't use malformed barycentric coordinates, once again the tetrahedron (P_d,P_0,P_1,P_2) is considered, and if any of the faces are obtuse then the method is switched to projecting P_d to the obtuse edge of the base triangle and calculating the coefficients necessary to linearly interpolate P_d from the edge points. If, however, P_d projects outside of the edge points then the nearest point in the triangle is used. Figure 3 contains pseudo code for calculating the coordinates.

```
void GeometryPartType::
UpdateBarycentricCoordinates(const VectorType&  Pd,
                             ElementType&       Element,
                             SearchResultsType& Results)
    {
    Float64 C[3];
    Float64 TotalArea;

    VectorType BaseNormal;
    VectorType Pm;

    VectorType& P0=Element.P0;
    VectorType& P1=Element.P1;
    VectorType& P2=Element.P2;

    BaseNormal=NormalToTriangle(P0,P1,P2);

    Pm=Pd-((Pd-P0)*BaseNormal)*BaseNormal; // Pd projected to the plain

    C[0]=AreaOfTriangle(P1,P2,Pm);
    C[1]=AreaOfTriangle(P2,P0,Pm);
    C[2]=AreaOfTriangle(P0,P1,Pm);

    TotalArea = C[0]+C[1]+C[2];

    C[0]/= TotalArea;
    C[1]/= TotalArea;
    C[2]/= TotalArea;
```

Figure 3. Pseudo Code for Calculating Barycentric Coordinates

```
      // Think of a tetrahedron Pd at the peak, P0,P1,P2 at the base
      // if the Pd projects into the base then the height will do.
      // if Pd projects outside the base then one of the faces will be
      // obtuse to the base, and the DistanceFromTheEdge will be
      // required.
      if (BaseNormal*NormalToTriangle(P0,P1,Pd)<0.0)  // if obtuse face
          {
          C[2]=0.0;
          UpdateBarycentricCoordinatesFromEdge(Pd,P0,P1,C[0],C[1]);
          }
      else if (BaseNormal*NormalToTriangle(P1,P2,Pd)<0.0)
          {
          C[0]=0.0;
          UpdateBarycentricCoordinatesFromEdge(Po,P1,P2,C[1],C[2]);
          }
      else if (BaseNormal*NormalToTriangle(P2,P0,Po)<0.0)
          {
          C[1]=0.0;
          UpdateBarycentricCoordinatesFromEdge(Po,P2,P0,C[2],C[0]);
          }
      Results.BarycentricCoordinates[0]=C[0];
      Results.BarycentricCoordinates[1]=C[1];
      Results.BarycentricCoordinates[2]=C[2];
      ];

  void GeometryPartType::
  UpdateBarycentricCoordinatesFromEdge(const VectorType& Po,
                                       const VectorType& P0,
                                       const VectorType& P1,
                                       Float64&          C0,
                                       Float64&          C1)
      {
      VectorType L =P1-P0;
      VectorType R0=Po-P0;
      VectorType R1=Po-P1;
      Float64    d;

      if      (R0*L<0.0)
          {
          C0=1.0;
          C1=0.0;
          }
      else if (R1*L>0.0)
          {
          C0=0.0;
          C1=1.0;
          }
      else
          {
          C1=(R0*Unit(L))/Abs(L);
          C0=1.0-C1;
          }
      };
```

Figure 3. (Continued)

Summary

An efficient method for unstructured mesh-to-mesh interpolation has been presented. This method uses a binary space partitioning tree to sort the elements of the source mesh. Using this tree data structure an efficient search is performed for the source mesh element nearest each of the destination mesh points. Once found the nearest source mesh element is used to compute barycentric coordinates which are then used as weighting coefficients for the interpolation. This method has been successfully implemented and used for coupling of a CFD and CSD code for the simulation of fluid structure interaction problems.

References

1. Headington, M. and R. Riley (1997). *Data Abstraction and Structures Using C++*, Jones and Bartlett Publishers, Sudbury, MA.
2. Samet, H. (2006). *Foundations of Multidimensional and Metric Data Structures*, Morgan Kaufmann, San Francisco, CA.
3. Coxeter, H. (1989). *Introduction to Geometry, 2nd Edition*, John Wiley & Sons, Inc., Hoboken, NJ.
4. Ericson, C. (2005). *Real Time Collision Detection*, Elsevier Inc., San Francisco, CA.

This page intentionally left blank.

Distribution

	Copies
Ki-Han Kim Code 333 Ballston Centre Tower One 800 North Quincy St. Arlington, VA 22217-5660	1
Captain Warren Defense Advanced Research Projects Agency 3701 N. Fairfax Dr. Arlington, VA 22210	1
Defense Technical Information Center 8725 John Kingman Road Suite 0944 Fort Belvoir, VA 22060-6218	1

NSWC, Carderock Div. Internal Distribution

Code	Name	Copies
3410	Smith, William	1
3452	TIC	1
5060	Walden, David	1
5700	Chang, Peter	1
5700	Ebert, Michael	1
5700	Kim, Sung-Eun	1
5700	Miller, Ronald	1
5800	Black, Scott	1
5800	Neely, Steve	1
5800	Hurwitz, Rae	1
6520	Thurman, Jeffery	1
7206	Madden, Craig	1